A PLATFORM FOR COLLABORATIVE ACOUSTIC SIGNAL PROCESSING

Hanbiao Wang[†], *Lewis Girod*[†], *Nithya Ramanathan*[†], *Deborah Estrin*[†], *Kung Yao*[♯]

University of California, Los Angeles
Computer Science Department[†] Electrical Engineering Department[♯]
Los Angeles, CA 90095
{hbwang, girod, nithya, destrin}@cs.ucla.edu[†], yao@ee.ucla.edu[♯]

ABSTRACT

In this paper, we present a platform for collaborative acoustic signal processing, and demonstrate its use with an example application. Our platform is built upon the Stargate Linux-based micro-server, and supports synchronized multi-channel acoustic data acquisition. We implement a dataflow-like staged event-driven programming model within the Emstar software framework that simplifies the development of collaborative processing applications. Unlike previous dataflow systems that emphasize real-time constraints, our framework emphasizes collaborative processing across nodes in a distributed system connected by an energy-conserving wireless network with non-deterministic message latency. In our model, an application is constructed by wiring together multiple stages, where each stage is implemented by an EmStar module. The modular approach simplifies development by isolating errors to specific stages, and enables run-time system reconfigurability by allowing users to swap out implementations of individual stages, and to reconfigure the dataflow at run time.

1. INTRODUCTION

The recent emergence of sensor network technology is driven by advances in a range of different disciplines. Collaborative signal and information processing is one application of sensor networks that shows great promise to reveal previously indiscoverable properties of physical phenomena. In contrast to global data collection, local collaboration and processing can achieve more with the same resources, by enabling higher sampling densities and by placing sensors closer to the phenomena under investigation. For example, collaborative acoustic signal processing can distill out information about the sound source and the propagation media from the raw data, reducing the incremental network cost to support each node. Because of its crucial role in many applications, much research effort has been devoted to collaborative processing algorithms in sensor networks [1, 2]. However, this research into algorithms has typically stopped short of addressing the practical ways in which platform design, system architecture, and programming models can facilitate the development and testing of collaborative processing in sensor networks.

In this paper, we present a platform for exploring collaborative acoustic signal processing in micro-server based sensor networks. Our platform supports a dataflow model within each node as well as collaboration across nodes, all built upon the EmStar software framework [3, 4]. We have extended the Stargate [5] platform, adding a synchronized four-channel acoustic data acquisition capability using the commercial-off-the-shelf (COTS) hardware and the open-source Linux based software.

Our platform's data-flow model is implemented by a staged event-driven programming framework designed to support collaborative acoustic signal processing applications. In this model, a collaborative processing application is divided into a sequence of stages. Each stage conditionally generates events to trigger the following stage if and only if the result of the current processing stage indicates that the observed phenomenon could be relevant to the sensing application goal. Irrelevant events are filtered at the earliest time, thus saving system resources. Each stage is implemented in a separate process as an individual Emstar module, thus isolating errors to the scope of individual stages.

In order to support the unique properties of sensor networks, we have designed this model to be much more loosely coupled than many previous dataflow system designs. Whereas most current dataflow implementations foster determinism in order to provably meet real-time requirements, this is often an impossibility in the context of sensor networks. In order to reduce the energy requirements of the wireless networks connecting sensor nodes, the network protocols employed often increase the variance of network latency to conserve energy. In the context of collaboration, this variance translates to signal buffering, because the importance of signal data may not be known until after a delayed message arrives. The decreased reliability of individual nodes and network links further exacerbates this problem, as protocols must often handle cases where some nodes fail to respond.

Given these relaxed timing requirements, our framework leverages this loose coupling to simplify application development and to build a more flexible and robust system. By implementing each stage as a separate EmStar module, we gain several advantages.

First, the system is both easier to debug and more robust, because numerical errors and other failures are isolated to specific stages, rather than occurring somewhere in a monolithic block. Robustness is particularly critical in deployment, because errors can be triggered by conditions that arise only in the field, such as unusual or unexpected sources of noise and system failures induced by a harsh environment. In our framework, EmStar can restart individual stages of the dataflow when they crash or hang, without disrupting the whole system.

Second, the system is much more flexible because of its modular construction. Because the dataflow "wiring" is determined at run-time, our model enables the system to switch between different implementations of the same stage after deployment, or even

This work is supported by the National Science Foundation (NSF) under Cooperative Agreement #CCR-0121778.

to completely change the data-flow wiring at run-time. Such reconfigurability helps the system adapt to changes in application requirements, signal characteristics, and physical environment.

In this paper, we present three main contributions. First, we present a hardware platform that integrates a Stargate processor module with a sound subsystem that captures sound synchronously through four channels and integrates to other EmStar services. Second, we present a staged event-driven programming model designed for collaborative processing applications that conserves network and system resources, eases collaborative processing programming, and enables run-time system reconfigurability. Third, we present an example collaborative signal processing application that we have built using this model and platform.

The rest of the paper is organized as follows. Section 2 describes the hardware and software building blocks of the platform for collaborative acoustic signal processing. Section 3 describes the staged event-driven programming model on top of Emstar. Section 4 illustrates the platform with an example of collaborative processing that localizes woodpeckers using sound. Section 5 compares our approach with traditional embedded signal processing systems. Section 5 also reviews existing work on collaborative processing in sensor network. Section 6 concludes this paper

2. BUILDING BLOCKS

This section describes the subsystem for synchronized multi-channel data acquisition and the Emstar software environment supporting the staged event-driven programming model.

2.1. Acoustic data acquisition

Our previous work in acoustic beamforming applications used COTS iPAQs [6, 7]. Using this platform, we were forced to use the iPAQ's built-in microphones and audio codecs, which provided a single channel with high internal noise and low sensitivity. However, this hardware is not sufficient to capture weak signals such as wild bird calls.

To address these limitations, we developed a new platform based on the Stargate processing module and a high-quality multichannel sound card and external microphones. The Stargate [5] is a micro-server class platform based on the 400 MHz PXA55 XScale processor with 64 MB of SDRAM, 32MB of flash memory, and provides similar processing capability to the iPAQ. To the Stargate we add the VXpocket 440 PCMCIA sound card [8] to support acoustic sampling, and a Compact Flash 802.11 card to support a wireless network. The VXpocket has 4 balanced mic/line analog inputs, provides 24-bit high-quality audio measurement, and a sampling frequency that ranges from 8kHz to 48 kHz in 100 Hz steps. We chose the M53 [9] for our low noise, low distortion measurement microphone. We connect the M53 to the VXpocket440 card using a cable that converts the TB3M connector of the M53 to a standard 3-pin male XLR connector of the VXpocket440. Each M53 microphone is calibrated using a free-field comparison procedure with a laboratory grade reference microphone. The calibration produces a precision error response curve which can be used for correcting the response.

The VXpocket card is supported by an Advanced Linux Sound Architecture (ALSA) [10] device driver, a user space library, and management tools. Compared to the Linux implementation of Open Sound System (OSS), ALSA has many advantages. ALSA's consistent and generic control API for managing low-level hardware controls enables a developer to easily take advantage of the advanced features of the sound cards. ALSA uses a ring buffer to store outgoing or incoming samples with two pointers to allow precise communication between application and sound card device. The hardware maintains the pointer to the most recently captured sample while the application maintains the pointer to the first unread sample. ALSA supports both standard read/write transfer for simplicity and direct read/write transfer via direct memory access (DMA) for efficiency. The ALSA API has event waiting routines and asynchronous notification handling constructs. Using ALSA, we have managed to synchronously sample sound on 4 input channels of the same VXpocket 440 sound card with a measured time synchronization discrepancy across input channels of about $2 \sim 5$ μs.

2.2. Emstar software environment

Emstar [3, 4] is a comprehensive software environment for developing heterogeneous, distributed applications. Emstar provides tools for simulation, emulation, and visualization of Emstar based distributed systems. It also provides many services such as networking and time synchronization across nodes. The strengths of Emstar lie in its message-passing inter process communication (IPC) primitives. Each Emstar based system consists of multiple logically separable modules implemented as individual processes. Modules communicate with one another using message-passing. Within this framework, modules are implemented and debugged separately, thus the application development becomes much easier.

Emstar's message-passing IPC is implemented in user space device drivers. The Framework for User Space Devices (FUSD) [4] consists of a kernel module and user space libraries. FUSD can create device files and then proxy system calls on FUSD device files into user-space device drivers. From the client's point of view, FUSD device files respond in the same way as ordinary device files semantically, although there is typically a performance penalty relative to an in-kernel driver. Compared to device drivers implemented in kernel, FUSD-proxied device drivers add the overhead of additional system calls, scheduling latency, and additional data copy. However, given ever-increasing hardware speed, it makes sense for Emstar to favor development flexibility over system performance, and only add optimizations as they are needed by applications. In collaborative signal processing applications, real-time performance is often not a practical objective. Because energy-conserving network communication often incurs nondeterministic latency, our systems are typically required to buffer data so that it can later be processed, after the network messages arrive. This relaxation of real-time requirements for collaborative processing applications reduces the importance of the additional latency introduced by FUSD message-passing.

On top of the message-passing IPC mechanism provided by FUSD devices, Emstar implements an event-driven programming model using the Glib event framework. Emstar modules that generate events create a FUSD device and then emit event messages through that device. Emstar modules that need to receive those events listen to that FUSD device, and are notified asynchronously when new event messages arrive. The Emstar libraries support several different types of of FUSD devices and their clients, many of which are customized to support specific classes of application with minimal effort. The details of creating the FUSD devices and

listening on file descriptors are all abstracted away from the application programmer.

The sensor device interface is a FUSD device specifically designed to support signal data streams and detection events in collaborative signal and information processing applications. The sensor device interface has two primary functions: buffering and time-stamping. The sensor device provides a ring buffer that stores a sequence of data samples or events and integrates with the Emstar time synchronization service to time-stamp the data elements. A client module of the sensor device driver can request data either synchronously or asynchronously. The client module can request a single sample from a specific time, a replay of data from a specific time interval, or a continuous stream of samples as they arrive. The sensor device interface provides the foundation for the staged event-driven collaborative processing model.

3. STAGED EVENT-DRIVEN MODEL

In order to support collaborative signal and information processing applications, we have developed a staged event-driven signal processing model implemented within the EmStar framework. This model is similar to the dataflow computation model in which a set of processing functions are connected by data queues. A processing function consumes data from its input queues and produces data in its output queues. Execution of a processing function starts if only if data is available at its input data queues. There is no central control, and all processing functions run autonomously in parallel. Data flow is a natural paradigm for building digital signal processing applications for concurrent implementation on parallel or distributed hardware. Its history goes back to the sixties when Estrin and Turn proposed an early dataflow model [11]. Over the years, many variants and extensions of the basic dataflow model have been developed [12, 13, 14].

In our implementation, we borrow the basic ideas of the dataflow computation model to organize collaborative processing applications in Emstar, while relaxing the requirement for real-time scheduling of processing functions. With message-passing IPC and an asynchronous event framework, Emstar naturally supports the basic dataflow computation model. Each processing function can be implemented as an Emstar sensor device driver module whose device ring buffer serves as output data queue. In order for a processing function to use a data queue for input, the Emstar module that implements the processing function requests data from the sensor device that implements the data queue.

In the staged event-driven programming paradigm, a collaborative processing application is decomposed into a sequence of processing stages. Signals enter the system from the network or through a data acquisition stage. Contrary to some variants of the dataflow model such as synchronous dataflow [13], the staged event-driven processing of incoming signals does not always reach the final stage. Rather, each processing stage will only generate an event to trigger the following processing stage if its result meets a specified condition.

This staged processing model is a natural fit for monitoring applications that would like to ignore many irrelevant events. For example, a system to identify and locate woodpeckers using sounds can be decomposed into 4 stages as shown in Fig. 1. The first stage samples sound, time-stamps samples, and buffers them. The second and third stages perform detection algorithms and generate an event to the following stage only if the signal is important to the application. The fourth stage, target localization, combines data from multiple nodes and multiple channels to localize the woodpecker, and provide that output to a higher layer application. By stopping the dataflow early, we save system resources that would otherwise be wasted in the following stages.

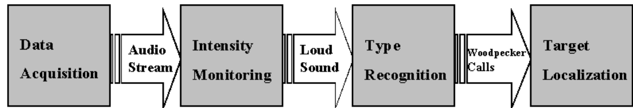

Fig. 1. Example block diagram of staged event-driven processing for woodpecker localization applications. Grey rectangles stand for processing stages. Block arrows represent asynchronous data or event queues implemented using sensor devices that connect adjacent processing stages.

The staged event-driven programming model improves utilization of system resources and eases system development. This model naturally decomposes a complex application into a set of simpler processing stages that are accurately specified by a block diagram and are readily allocated among a team of developers. It also enables individual processing stages to be independently implemented and debugged, since each processing stage is incorporated inside an Emstar module and is executed as an autonomous process. Compared to a monolithic architecture, the staged evendriven framework greatly simplifies the implementation of a collaborative processing application, and results in a more robust solution. Using the staged even-driven framework, numerical errors only affect a single module, and thus are easier to trace. In the event that a numerical bug is uncovered in a deployment, the system can kill and restart the misbehaving stage without causing the whole system to fail.

The staged event-driven framework in Emstar also enables system reconfigurability. Given a processing task, there are often multiple processing algorithms that realize it. For example, there are many different algorithms to estimate target location using sound, which have different characteristics in terms of signal requirements, computation complexity, and estimation error. For the same processing stage, we can implement multiple versions using different processing algorithms. Since all of the different implementations of the same processing stage use the same data input and output interfaces, we can select an implementation at run-time based on application requirements, system resources, and even dynamic signal characteristics.

4. CASE STUDY

In this section, we describe the implementation of the staged event-driven processing in a simple sensor network using two nodes to localize a woodpecker using acoustic signals. The system is schematically shown in Fig. 2. Each node has four microphones to capture the sound synchronously. Using acoustic beamforming, each node can individually estimate the Direction of Arrival (DOA) of sound from the target. The target location can be estimated as the intersection of these two bearing lines. We designate bearing crossing task to node A. Node B needs to send its DOA estimate to node A. We intentionally make the scale of the system small for simplicity of illustration. We also intentionally gloss over the signal processing algorithm details in order to emphasize the software system architecture.

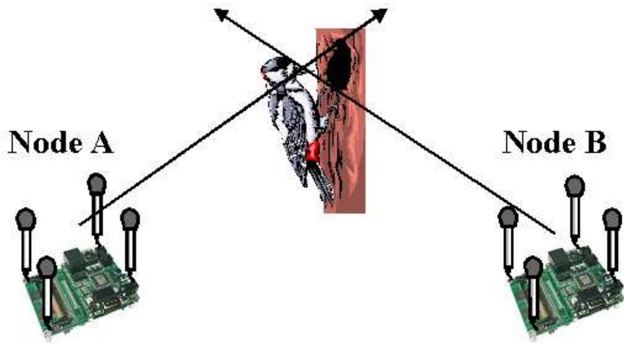

Fig. 2. Schematic diagram of a simple network of two nodes for localizing a woodpecker using sound. Each node has four microphones to capture sound synchronously. DOA pointing to the woodpecker can be estimated by individual node using acoustic beamforming. Arrows indicated individual DOA estimates. The location of the woodpecker can then be estimated as the intersection of these two bearing lines. If the bearing crossing task is designated to node A, the node B needs to send its own DOA estimate to node A.

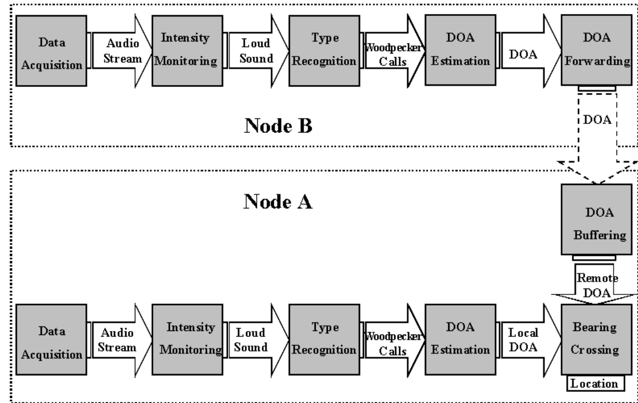

Fig. 3. Block diagram of staged event-driven processing for woodpecker localization sensor network as shown in Fig. 2. Grey rectangles stand for processing stages. White rectangles indicate device interfaces. Block arrows represent data or event queues. Subsets of block diagrams physically located on each individual node are enclosed by dotted rectangles. The dotted block arrow represents the DOA message passing channel from node B to node A.

As shown in Fig. 3, the collaborative processing in the two-node sensor network is decomposed into a sequence of simpler processing stages, in which each stage is driven by data or events from the proceeding stage. On each node, the first stage samples sound, time-stamps the samples, and buffers them. The second stage, "Intensity Monitoring", monitors signal intensity, and generates an event to the next stage only if a strong signal intensity indicates that the signal is not just background noise. The third stage, "Type Recognition", attempts to identify the captured sound to determine if it was produced by a woodpecker, and generates an event only if there is an above-threshold likelihood that the sound was the call of a woodpecker. The fourth stage, "DOA Estimation", combines data from multiple channels to determine a bearing estimate towards the woodpecker, and generates an event only if a valid bearing can be determined.

Valid bearing estimates captured at node B are forwarded over the network to node A where they are combined in a "Bearing Crossing" stage to localize the woodpecker. This stage buffers bearing estimates from many nodes in order to support high variance in the arrival times of the estimates over the network, and correlates them using the EmStar time synchronization subsystem. When a location estimate is determined it is provided through another device interface to a higher-level application.

Without staged filtering of irrelevant events, the system would be overwhelmed by irrelevant events most of the time, and would unnecessarily consume scarce CPU and network resources. However, many stages of our implementation cuts short processing on signals that are not important to our application. Most of the time, the raw captured sound is just background noise and will be filtered out by early stages. Low amplitude signals will be filtered by the Intensity Monitoring stage, while loud signals that are not bird calls will be filtered by the Type Recognition stage. Incoherent data that cannot generate a bearing estimate will be filtered by the DOA Estimation stage, saving both CPU and network resources. By terminating processing early, we save system resources that would otherwise be wasted in following stages.

In building this application we took advantage of many of the benefits intrinsic to this platform and framework. The modular design of our framework enabled us to build and test this system incrementally stage by stage. During development, we experienced many numerical errors such as overflow and underflow. Because our application was broken into individual stages, the scope of error tracing was limited and debugging time was greatly reduced.

We also made extensive use of our framework's capacity for system reconfigurability. Our implementation of the data acquisition stage provides two modes: deployment and simulation. The deployment mode directly captures sound through the sound card and microphones, while the simulation mode plays back previously recorded bird calls. To test the system's other processing stages in the lab without the sound subsystem, we simply switched to the simulated data acquisition mode, keeping all other processing stages the same.

We were also able to use this modularity to test our system using two alternative implementations of the intensity monitoring stage. One is based on a specified threshold of data sample value, and the other is based on Constant False Alarm Rate (CFAR) detection of signal power in the frequency bandwidth of woodpecker sound. The former implementation is much simpler and runs much faster than the latter. However, the latter differentiates meaningful sound from background noise much more accurately in terms of false alarm rate and miss rate. Our framework allows us to easily switch between these two types of implementation without modifying anything else in the system. This reconfigurability makes it easier to compare these two approaches for signal intensity monitoring under the same condition.

Similarly, we can easily test multiple implementations of the type recognition stage and the DOA estimation stage. Our current implementation of the type recognition stage is based on cross-correlation of the measured spectrogram and the reference spectrogram [7]. Our current DOA estimation implementation is based on approximate maximum likelihood (AML) beamforming algo-

rithm [15, 16]. Implementation based on other algorithms can be chosen for the system to adapt to variations of the signal characteristics, the system resources, and the application requirements.

5. DISCUSSION

This section reviews the traditional embedded system for signal processing and the recent work on programming collaborative processing in sensor networks.

Research on embedded system for signal processing and control has a long history. Over the years, many powerful frameworks and tools have been invented for specification, modeling, simulation, verification, and code generation of signal processing embedded system. For example, several variants of the dataflow model [12, 13, 14] with formal mathematical properties have been proposed. Thus important questions about the system behavior can often be answered, e.g. whether the computation executes in finite memory and finite time. Another example is MathWorks Simulink product family [17] that enables designers to apply streamlined model based design in a graphical, interactive environment. The design starts from a system-level mathematical model and then is tested and verified with simulation. Implementation code can be generated automatically. Yet another example is Ptolemy system-level design tool [18, 19, 20] that seamlessly supports multiple computational models using actor-oriented design. There are also many specially designed high-level programming languages to simplify digital signal processing. Examples include Disiple [21], DSP/C [22], DSPL [23], Gabriel [24], and GOSPL [25]. These languages use compiler techniques to target specific hardware architecture or to realize real-time scheduling. Limited space does not allow us to describe other important work in the traditional embedded system field.

There are many significant differences between the traditional embedded system for signal processing and our work on collaborative processing in sensor networks. First the traditional design methodologies and tools often utilize specially designed platforms. Our Emstar based staged event-driven programming paradigm targets general purpose micro-server class platforms running open-source Linux. Second the traditional design methodologies often have formal mathematical properties. They can be used to analyze system behavior based on system mathematical model without implementation. Our staged event-driven programming frame work is only meant for fast prototyping and testing collaborative processing applications in a distributed systems. Most importantly, the traditional embedded system design supports real-time computation scheduling while our programming framework does not support hard realtime applications. Process scheduling in Linux on Stargate has a coarse scheduling granularity of 10 ms. Scheduling delay can also be nondeterministic. In addition, message passing has nondeterministic delay, especially when message has to cross individual nodes through network using contention based media access control (MAC) such as 802.11.

Although our staged event-driven programming framework provides best-effort service without a realtime guarantee, it is a very good fit for fast prototyping and testing collaborative processing applications in sensor networks. First we use time-stamping and buffering of data to overcome the difficulties of nondeterministic delays. Time-stamping and buffering of data enable post-facto processing of data that removes negative effects of coarse scheduling granularity and nondeterministic delay largely. Second our programming framework does not add more to the constraints imposed by contention based MACs on sensor networks. Using contention based MACs, it is inherently not possible to guarantee message delivery within a certain deadline. As a result, collaborative processing applications in such sensor networks cannot impose realtime deadlines. In addition, the general-purpose hardware and open-source Linux used in our testbed are more accessible to the sensor network research community than specially designed hardware and commercial system software.

Although active work on programming in sensor networks exists, it has not provided adequate programming support to develop collaborative processing applications. TinyGALS is a globally asynchronous and locally synchronous model for event-driven embedded systems [26]. Its code generation tools have been implemented for Berkeley motes [27]. Unfortunately, the current generation of motes do not have sufficient system resources to support coherent processing of multi-channel, high-rate signals such as acoustic beamforming. An information directed approach is proposed to coordinate collaborative signal and information processing tasks in sensor networks [28]. However, it has not yet touched upon the programming issues in sensor networks. DFuse is a framework for distributed data fusion in sensor networks [29]. Its API enables an fusion application to be specified as a course-grained dataflow graph. It emphasizes the dynamic assignment of aggregation roles to sensors at run time for energy efficiency. Visual Sense is a modeling and simulation framework for wireless sensor networks [30]. Built on top of Ptolemy II [18, 19, 20], it supports the actor-oriented computing model. However, it does not provide the software environment to ease the implementation of collaborative processing applications.

The simple example of a woodpecker localization system demonstrates the strengths of the staged event-driven programming framework. Many other previously proposed collaborative processing systems can also benefit from this programming framework for ease of implementation, enhanced utilization of system resources, and reconfigurability. For example, the collaborative energy-based localization scheme proposed by Sheng and Yu [31] can be naturally implemented using this programming framework. Another example is the collaborative classification scheme proposed by D'Costa and Sayeed [32]. Depending on correlation in measurements, either data fusion or decision fusion is used for classification in this scheme. Our programming framework naturally introduces reconfigurability for the above classification scheme.

6. CONCLUSION AND FUTURE WORK

In this paper, we have described a platform for prototyping collaborative acoustic signal processing applications on micro-server platforms running Linux. We have also presented an example of collaborative signal processing application built using this platform. First, we have realized synchronized acoustic data acquisition on the Stargate platform within the Emstar software environment. Second, we have implemented a dataflow-like staged event-driven programming model in Emstar that simplifies the development of collaborative processing applications. The model also enables run-time system reconfigurability by allowing users to swap out alternative implementations of individual stages, or to reconfigure the dataflow at run time. In addition, the model improves system resource utilization by filtering out irrelevant events as early as possible.

We plan to continue development of this system to support numerous collaborative signal processing applications at CENS. As

we develop new applications, we will profile the system to discover bottlenecks and to determine which optimizations to make first. Some optimizations and future directions include:

- Develop an efficient shared-memory mode for our sensor device pattern.
- Apply realtime Linux features such as fine-grained scheduling, time interval and priority based scheduling.
- Automate the selection of stage output queue sizes, to minimize memory usage.
- Offload computation-intensive processing stages to external processors while retaining the same staged event-driven programming paradigm.

7. REFERENCES

[1] S. Kumar, D. Shepherd, and F. Zhao, "Collaborative signal and information processing in microsensor networks," *IEEE Signal Proc. Mag.*, vol. 19, no. 2, pp. 13–14, March 2002.

[2] K. Yao, D. Estrin, and Y.H. Hu, "Sepcial issue on sensor networks," *EURASIP JASP: Special Issue on Sensor Networks*, vol. 2003, no. 4, pp. 319–320, March 2003.

[3] EmStar: Software for Wireless Sensor Networks, "http://cvs.cens.ucla.edu/emstar/," 2004.

[4] L. Girod, J. Elson, A. Cerpa, T. Stathopolous, N. Ramanathan, and D. Estrin, "Emstar: a software environment for developing and deploying wireless sensor networks," in *Proc. USENIX'04 Annual Tech. Conf., General Track*, Boston, June 2004, pp. 283–296.

[5] Platform X with Stargate, "http://platformx.sourceforge.net/links/resource.html," 2004.

[6] H. Wang, L. Yip, D. Maniezzo, J. C. Chen, R. E. Hudson, J. Elson, and K. Yao, "A wireless time-synchronized cots sensor platform, part ii: applications to beamforming," in *Proc. IEEE CAS Workshop on Wireless Communication and Networking*, Pasadena, CA, USA, September 2002.

[7] H. Wang, J. Elson, L. Girod, D. Estrin, and K. Yao, "Target classification and localization in habitat monitoring," in *Proc. ICASSP*, Hong Kong, China, April 2003.

[8] Diggigram VXpocket 440 PCMCIA Sound Card, "http://www.digigram.com/products," 2004.

[9] M53 Microphone: Low Noise Low Distortion Measurement Microphone, "http://www.linearx.com/products/microphones /m53/m53_1.htm," 2004.

[10] The Advanced Linux Sound Architecture (ALSA), "http://www.alsa-project.org/," 2004.

[11] G. Estrin and R. Turn, "Automatic assignment of computations in a variable structure computer system," *IEEE Transactions on Electronic Computers*, vol. 12, no. 6, pp. 755–773, December 1963.

[12] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing74: Proceedings of IFIP Congress*, pp. 471–475, August 1974.

[13] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *Proc. of IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept. 1987.

[14] M. Engels, G. Bilsen, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," in *Asilomar Conf. Sig. Sys. and Comp.*, Pacific Grove, California, Oct. 1994.

[15] J. Chen, L. Yip, J. Elson, H. Wang, D. Maniezzo, R. Hudson, K. Yao, and D. Estrin, "Coherent acoustic array processing and localization on wireless sensor networks," *Proc. the IEEE*, vol. 91, no. 8, pp. 1154–1162, August 2003.

[16] P. Bergamo, S. Asgari, H. Wang, D. Maniezzo, L. Yip, R.E. Hudson, K. Yao, and D. Estrin, "Collaborative sensor networking towards real-time acoustical beamforming in free-space and limited reverberance.," *IEEE Trans. Mob. Comput.*, vol. 3, no. 3, pp. 211–224, 2004.

[17] The MathWorks Products, "http://www.mathworks.com," 2004.

[18] E.A. Lee, "Overview of the ptolemy project," Technical Report UCB/ERL M03/25, University of California, Berkeley, 2003.

[19] E.A. Lee, *Embedded Software*, vol. 56, chapter Advances in Computers, Academic Press, London, 2002.

[20] E.A. Lee and S. Neuendorffer, *Actor-oriented Models for Codesign*, chapter Formal Methods and Models for System Design, Kluwer, 2004.

[21] J.E. Peters and S.M. Dunn, "A compiler that easily retargets high level language programs for different signal processing architectures," in *Proc. ICASSP*, May 1989, pp. 1103–1106.

[22] K. Leary and W. Waddington, "Dsp/c: a standard high level language for dsp and numeric processing," in *Proc. ICASSP*, Apr. 1990, pp. 1065–1068.

[23] A. Schwarte and H. Hanselmann, "The programming language dspl," in *Proc. PCIM'90*, 1990.

[24] E.A. Lee, E. Goei, J. Bier, and S. Bhattacharya, "A design tool for hardware and software for multiprocessor dsp systems," in *Proc. ISCAS'89*, 1989.

[25] C.D. Covington, G. Carter, and D. Summers, "Graphic oriented signal processing language - gospl," in *Proc. ICASSP*, 1987, pp. 1879 – 1882.

[26] E. Cheong, J. Liebman, J. Liu, and F. Zhao, "Tinygals: a programming model for event-driven embedded systems," in *Proc. SAC'03*, Melbourne, FL, March 2003, pp. 698–704.

[27] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.E. Culler, and K. Pister, "System architecture directions for network sensors," in *Proc. ASPLOS'00*, Cambridge, MA, Nov. 2000.

[28] F. Zhao, J. Shin, and J Reich, "Collaborative signal and information processing: an information-directed approach," *Proc. IEEE*, vol. 91, no. 8, pp. 1199–1209, August 2003.

[29] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin P. Hutto, A. Paul, and U. Ramachandran, "Dfuse: a framework for distributed data fusion," in *Proc. SenSys'03*, Los Angeles, CA, Nov. 2003, pp. 114–125.

[30] P. Baldwin, S. Kohli, E.A. Lee, X. Liu, and Y. Zhao, "Modeling of sensor nets in ptolemy ii," in *Proc. IPSN'04*, Berkeley, California, USA, April 2004, pp. 359–368.

[31] X. Sheng and Y.H. Hu, "Energy based acoustic source localization.," in *Proc. IPSN*, 2003, pp. 285–300.

[32] A D'Costa and A M. Sayeed, "Collaborative signal processing for distributed classification in sensor networks.," in *Proc. IPSN*, 2003, pp. 193–208.